AD-A068 847 WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER F/G 9/2
TRIPLEX ARITHMETIC FOR FORTRAN.(U)
DEC 78 K BOEHMER, J M YOHE DAAG29-75-C-0024
UNCLASSIFIED MRC-TSR-1901 NL

1 OF 1
ADA
068847

END
DATE FILMED
6-79
DDC

MRC Technical Summary Report #1901

TRIPLEX ARITHMETIC FOR FORTRAN

Klaus Boehmer and J. M. Yohe

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

See back page for 1473

December 1978

Received October 26, 1978

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

Institut fuer Praktische Mathematik
Universitaet Karlsruhe
D-7500 Karlsruhe, Germany

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

TRIPLEX ARITHMETIC FOR FORTRAN

Klaus Boehmer (1) and J. M. Yohe (2)

Technical Summary Report #1901
December 1978

ABSTRACT

Triplex arithmetic is a variant of interval arithmetic in which a "main" value is computed in addition to the endpoints of the containing interval. The "main" value is, in general, the value that would have been computed had properly-rounded real arithmetic been used to compute the results of the calculation; it may, in some sense, be regarded as the "most probable" value of the result of the calculation. The endpoints of the interval define the worst-case range of the possible values of the computation.

This report describes a FORTRAN implementation of triplex arithmetic in both single and multiple precision. The package described in this report is designed to be used with the AUGMENT precompiler; this makes triplex arithmetic particularly easy to use.

AMS(MOS) Subject Classification: 68A10

Key words: Interval Analysis
Interval Arithmetic
Multiple Precision
Extended Precision
Portable Software
Software Package
Precompiler Interface
Augment Interface
Triplex Arithmetic

Work Unit Number 8 (Computer Science)

(1) Institut fuer Praktische Mathematik
Universitaet Karlsruhe
D-7500 Karlsruhe, Germany
(2) Mathematics Research Center
University of Wisconsin - Madison
Madison, Wisconsin 53706

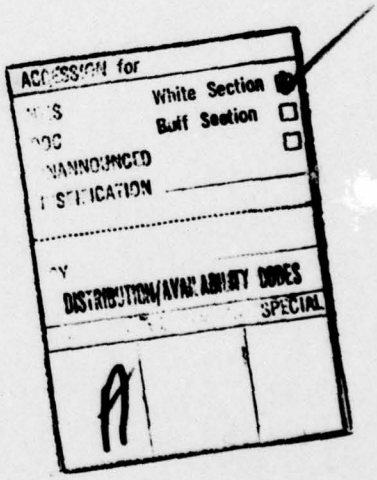
Sponsored by the U. S. Army under Contract No. DAAG29-75-C-0024, the Deutsche Forschungsgemeinschaft under contract No. BO 622/1, and the University of Karlsruhe.

SIGNIFICANCE AND EXPLANATION

In some computations, it is essential to determine rigorous bounds on the computed results. Interval arithmetic can be used for this purpose; the endpoints of the computed intervals give rigorous bounds on the range of the result values. One of the difficulties with interval arithmetic is that the intervals may become so wide as to be meaningless in some problems. One might elect to take the midpoint of the result interval as being an approximation to the value of the result, but this may not be entirely satisfactory, either -- for example, if the result interval were [-infinity, infinity] the midpoint would be computed to be zero -- but this is hardly a meaningful result, since the midpoint could be any real number.

In triplex arithmetic, a "main" value is computed; this value is the result that would have been obtained had properly-rounded real arithmetic been used in the computation. This gives, in some sense, a "most probable" value for the result.

In this paper, the authors describe a program package which performs calculations in triplex arithmetic. The package is written, insofar as practicable, in ANSI standard FORTRAN, and it is designed to be used in conjunction with the AUGMENT precompiler for FORTRAN to afford the user a relatively painless way of carrying out computations in Triplex arithmetic.



The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the authors of this report.

TRIPLEX ARITHMETIC FOR FORTRAN

Klaus Boehmer and J. M. Yohe

1. Introduction:

Interval arithmetic is a means of obtaining rigorous bounds on the results of computations by carrying both upper and lower bounds through the computation. This method was first formalized by R. E. Moore [11], and is discussed in detail there and elsewhere ([1], [12]). Briefly, however, the bounds obtained by interval computations represent worst-case bounds for all facets of the computation: error of approximation in the input data, error in obtaining the input data (via measurement, for example), and errors introduced in the computation.

One of the difficulties with interval arithmetic is that the intervals may become so wide as to be meaningless in some problems. One might elect to take the midpoint of the result interval as being an approximation to the value of the result, but this may not be entirely satisfactory. Let us suppose, for example, that we are working with a 6-place decimal computer. Using interval representation, we would write

$$\Pi = [3.14159,\ 3.14160].$$

Note that $\Pi = 3.1415926535\ldots$ belongs to the interval Π. Π is a pretty good approximation; however, we would find that

$$\Pi^2 = [9.86958,\ 9.86966]$$

-- a spread of 8 in the last position. If we take the midpoint of this, we have

$$\pi^2 \simeq 9.86962$$

-- which does not compare favorably with the correctly rounded value of $\pi^2 = 9.8696044\ldots$. We would in fact obtain 9.86959 if we were to apply correct rounding to the product 3.14159 x 3.14159. The reason for this is that often the round-off errors in computations cancel one another in a very advantageous way, so a result obtained by such a correct round-off procedure is often much better than the endpoints, or even the midpoint, of the computed interval.

Therefore, it has sometimes been proposed that an interval be represented as a midpoint together with an error term e (this is known as the centered form). In theory, this is just as good as the representation in terms of

---

Sponsored by the U. S. Army under Contract No. DAAG29-75-C-0024, the Deutsche Forschungsgemeinschaft under contract No. BO 622/1, and the University of Karlsruhe.

endpoints; in practice, it is not. If we were to represent Π in centered form, we would need to write

$$\Pi = 3.14159 \pm .00001$$

This would give us

$$\Pi^2 = 9.86959 \pm .00007,$$

generating an upper bound which is no better and a lower bound which is worse. (We acknowledge that, in the case of high precision calculations, the additional error introduced by this method is not so serious, and considerable storage economy may be realized by using the centered form. This was discussed by Aberth in [1] and in [2].)

We also acknowledge that it is possible to gain extra precision in the centered form by eliminating leading zeros in the error term, thus, for example, representing Π by

$$\Pi = 3.14159 \pm 2.65359 \times 10^{-6}.$$

In this case, we would obtain

$$\Pi^2 = 9.86959 \pm 1.89449 \times 10^{-5}$$

for a low-order spread of about 3.8. This is more accurate than standard interval arithmetic, but requires additional computing effort which might not be realizable or practical on a given computer.

Alternatively, we may carry three numbers through the computation: upper and lower bounds, as for intervals, and a main value, which represents the correctly rounded result of the computation. Using this scheme, we would have the triplex numbers

$$\Pi^* = [3.14159, 3.14159, 3.14160],$$

and

$$\Pi^{*2} = [9.86958, 9.86959, 9.86966].$$

In this case, the main value is very close to the correctly rounded value of the result. Obviously, it need not be so, and the only interpretation that can be given to the "main" value is that it is "probably" "close" to the correct answer.

This, then, is the motivation for triplex arithmetic. If we were not concerned with finite precision calculation, we would not need to distinguish between triplex, the centered form, and interval endpoint representation of data, since they would all give the same result. In the context of fixed finite precision representation, however, triplex can provide us with additional insight.

The theory of triplex arithmetic is developed extensively in [11] and [12], for example, and we refer the reader to these sources for a more comprehensive discussion.

The first complete system for performing triplex computations that is known to us was the Triplex-ALGOL-60 compiler developed by Wipperman and others at the University of Karlsruhe [15] in the late 1960's. A previous triplex arithmetic package for FORTRAN was developed at the Mathematics Research Center, University of Wisconsin - Madison, by Boehmer and Jackson [2] in about 1975; this package comprised a complete single precision version (based on an earlier version of the interval arithmetic package; see [4]) and a multiple precision version with limited capabilities. Both versions were, however, heavily dependent on the UNIVAC 1100 architecture, and consequently lack the portability and flexibility of the present package.

In this paper, we describe a complete triplex arithmetic package for FORTRAN, based on the interval arithmetic package developed by the first author ([18], [17]). This triplex package comes in two versions: a single precision package which is reasonably portable (see [18]), and a completely portable multiple precision version based on the multiple precision arithmetic package of Brent [3].

Since there are strong similarities between this triplex package and the interval arithmetic package described in [17], we will not reformulate all of the presentation given there, but will refer to that paper where it seems appropriate.

The current package is designed to be used with the AUGMENT precompiler for FORTRAN (see [5], [6], [7]), and indeed is itself written with the aid of the AUGMENT precompiler. This is discussed more fully elsewhere ([8], [17]), and will not be iterated here. Of course, for the multiple precision version, an AUGMENT interface for Brent's package must be used; this is described in [4].

2. A formal definition of triplex numbers:

A triplex number is an ordered triple

$$T := (t_L, t_M, t_R) \quad \text{with } t_L \leq t_M \leq t_R \tag{2.1}$$

We call t_L the left endpoint, t_M the main value, and t_R the right endpoint of the triplex number.

We denote the interval defined by T to be

$$IT := [t_L, t_R] \tag{2.2}$$

so all three components of the triplex number are elements of the interval IT.

If $t_L = t_R$ (= t_M by definition) we say that T is degenerate. Degenerate triplex numbers may be identified with the corresponding real numbers; hence the reals may be regarded as a subset of the triplex numbers.

We want to point out, however, that the conversion of a real number $\underline{t}$ on computer in- or output does give an approximation for an unknown real number t, usually without any information about how well or how badly $\underline{t}$ approximates $t \in R$. If we find on output a degenerate triplex $[\underline{t}, \underline{t}, \underline{t}]$, approximating t, we know that $\underline{t} = t$ (see below). So, at least in this connection, there is much more information in a degenerate triplex number than in the corresponding real.

In finite precision calculations the situation is complicated by the need for rounding. We first have to presume that rounding is performed according to the mathematical foundation laid down by Kulisch [9]. If M is the set of machine numbers, $\phi: R \to M$ is said to be an optimal rounding if and only if

$$x \cdot \leq y \Rightarrow \phi(x) \leq \phi(y), \text{ and} \qquad \phi(x) = x \text{ for all } x \in M \tag{2.3}$$

Equation (2.3) implies that any real number which is not a machine number must be rounded to one of the two machine numbers which bracket it:

Lemma: If ϕ is an optimal rounding, and if

$$m_1 := \max \{\underline{x} \mid \underline{x} \in M, \underline{x} \leq x\} \leq x < m_2 := \min \{\underline{x} \mid \underline{x} \in M, \underline{x} \geq x\},$$

then $\phi(x) = m_1$ or $\phi(x) = m_2$.

Proof: We have $\phi(m_1) \leq \phi(x) \leq \phi(m_2)$ by (2.3a). But by (2.3b), $\phi(m_1) = m_1$, and $\phi(m_2) = m_2$, so $m_1 \leq \phi(x) \leq m_2$. Since $\phi(x) \in M$ and $M \cap [m_1, m_2] = \{m_1, m_2\}$, we have $\phi(x) = m_1$, or $\phi(x) = m_2$. []

Now we define a correct rounding ϕ of a triplex number into the set T_M of machine triplex numbers:

$\nabla(x)$ is the "optimal downward directed rounding", i.e.

$$\nabla(x) := \max \{\underline{x} \mid \underline{x} \in M, \underline{x} \leq x\} \quad \text{for all } x \in R, \tag{2.4}$$

$\Delta(x)$ is the "optimal upward directed rounding", i.e.

$$\Delta(x) := \min \{\underline{x} \mid \underline{x} \in M, \underline{x} \geq x\} \quad \text{for all } x \in R. \tag{2.5}$$

If ρ is an optimal rounding, we define

$$\phi(T) := \phi(t_L, t_M, t_R) := (\nabla t_L, \rho t_M, \Delta t_R) \in T_M \tag{2.6}$$

and obtain

Theorem: If T is a triplex number, then $\phi(T)$ is also a triplex number.

Proof: We must show that $\nabla t_L \leq \rho t_M \leq \Delta t_R$. With $m_1 := \max \{\underline{x} \mid \underline{x} \in M, \underline{x} \leq t_L\} \leq t_M$ we find by (2.4), (2.3b) and (2.3a)

$$\nabla t_L = m_1 = \rho m_1 \leq \rho t_M .$$

Quite similarly, we have with $m_2 := \min\{\underline{x} \mid \underline{x} \in M, \underline{x} \geq t_R\} \geq t_M$ the relation

$$t_M \leq m_2 = m_2 = \Delta t_R . \qquad []$$

Remark: A similar argument will show that a triplex number will round to a machine triplex number whenever the set of machine numbers into which the main value is rounded contains the set of machine numbers into which the endpoints are rounded. Thus we may use higher precision for the main value than for the endpoints if we so choose; however, we must not use lower precision (it is easy to construct a counterexample).

Thus, if we wish to use the centered form (as indicated above) to gain precision, we must regard the main value as having at least the same precision as the error term, with as many trailing zeros as necessary, and any nonzero digits beyond the nominal precision of the main value must be absorbed into the error term prior to rounding the triplex number to a machine triplex number.

In practice, we will consider only triplex numbers of the form (2.1), and we will use the rounding defined by (2.3), with ρ taken to be rounding to the nearest machine number.

3. Basic operations with triplex numbers

We have seen in Section 2 that a triplex number may be interpreted as a closed interval with a special element of this interval being labeled as the main value. Therefore, the operations which we now define for triplex numbers are all obtained from the corresponding operations for intervals and an additional operation for the main value.

If $*$ is one of the four standard arithmetic operators $+$, $-$, $\times$, $/$, and $T := (t_L, t_M, t_R)$ and $W := (w_L, w_M, w_R)$ are triplex numbers, then $T * W$ is defined to be the triple

$$T * W := (\inf \{t * w \mid t \in IT, w \in IW\}, t_M * w_M, \sup \{t * w \mid t \in IT, w \in IW\}) \text{ (provided } 0 \notin W \text{ for division)}. \tag{3.1}$$

This definition may be re-interpreted as

$I(T * W) = \{t * w \mid t \in IT, w \in IW\} = IT * IW$, and we have

$t_M * w_M \in I(T * W)$, since $t_M \in IT$, $w_M \in IW$,

with the known interval operation $IT * IW = I(T * W)$.

More generally, if f is a real-valued function of k real variables, which is defined and continuous on $IT_1 \times IT_2 \times \cdots \times IT_k$, we define

$$F(T_1, T_2, \cdots, T_k) := (\inf\{f(t_1, t_2, \cdots, t_k \mid t_i \in IT_i\},$$
$$f(t_{1M}, t_{2M}, \cdots, t_{kM}),$$
$$\sup\{f(t_1, t_2, \cdots, t_k \mid t_i \in IT_i\}) \qquad (3.2)$$

Again, (3.2) may be interpreted with the known united interval extension F of f as

$$I(F(T_1, T_2, \cdots, T_k) = F(IT_1, IT_2, \cdots, IT_k) :=$$
$$\{f(t_1, t_2, \cdots, t_k) \mid t_i \in IT_i\}, \text{ and we have}$$
$$f(t_{1M}, t_{2M}, \cdots, t_{kM}) \quad F(IT_1, IT_2, \cdots, IT_k) \text{ since } t_{iM} \in IT_i.$$

Thus, (3.1) and (3.2) define operations on triplex numbers.

In both definitions there is, as in the corresponding interval operations, the unpleasant task of computing the inf and the sup. Compactness arguments show that these values may be obtained as function values of certain points in IT x IW or $IT_1 \times IT_2 \times \cdots \times IT_k$, respectively. A complete discussion for the four arithmetic operations and for the elementary transcendental functions is given in [17]; several relational operators are also discussed there.

We also need operators which form triplex numbers from reals or vice-versa, give information about the structure of the interval defined by the triplex number, and evaluate certain logical functions.

Forming new triplex numbers:

Compose is a function of three real variables giving, in this order, the left endpoint, the main value, and the right endpoint of the triplex number.

Union is a function of two triplex variables yielding the triplex number T defined by $t_L := \min\{t_{1L}, t_{2L}\}$, $t_M := (t_{1M} + t_{2M}) / 2$, $t_R := \max\{t_{1R}, t_{2R}\}$. Thus, IT is the smallest interval containing $IT_1 \cup IT_2$.

Intersect is a function of two triplex numbers T_1, T_2 yielding the triplex T with $t_L := \max\{t_{1L}, t_{2L}\}$, $t_R := \min\{t_{1R}, t_{2R}\}$, $t_M := (t_L + t_R) / 2$, if $t_L \leq t_R$. For $t_L > t_R$, Intersect is not defined. Thus, when it is defined, IT is the largest interval in $IT_1 \cap IT_2$.

Measures of triplex numbers: These are obtained as the measures of the corresponding intervals as defined by Ris [14] and used in [17]. We have, in particular, Absolute Value of T, Length of T, Half-length of T, Midpoint of T, Size of T, Supremum of T, Infimum of T, Magnitude of T, Mignitude of T, Pivot of T, Sign of T, and the distance between two triplex numbers, $\text{dist}(T, W) := \max\{|t_L - w_L|, |t_M - w_M|, |t_R - w_R|\}$.

Logical operators: These provide the means to test various properties of triplex numbers.

Bad is a logical function of one triplex number. Its value is true if and only if T is an improper triplex number, that is, if $t_L > t_M$ *or* $t_R < t_M$.

OK is the logical inverse of Bad; i.e., its value is true if and only if T is a proper triplex number -- that is, if $t_L \leq t_M \leq t_R$.

Element of is a logical function of a real number x and a triplex number T. Its value is true if and only if x is an element of IT.

Subset (Superset) is a logical function of two triplex numbers T_1 and T_2. Its value is true if and only if IT_1 is contained in IT_2 (IT_2 is contained in IT_1).

One main point in implementing the definitions stated here is to take account of the effects of rounding. In Section 2, we have already pointed out that this rounding has to be done so as to ensure that the left endpoints are rounded downward, the right endpoints are rounded upward, and the main values are rounded to the nearest machine number. If F is a triplex function as defined in (3.2), therefore, we obtain the rounded value as

$$\begin{aligned} \underline{F}(T_1, T_2, \bullet\bullet\bullet, T_k) := \phi(F(T_1, T_2, \bullet\bullet\bullet, T_k) := \\ (\nabla\inf\{f(t_1, t_2, \bullet\bullet\bullet, t_k) \mid t_i \in IT_i\}, \\ \rho f(t_{1M}, t_{2M}, \bullet\bullet\bullet, t_{kM}), \\ \Delta\sup\{f(t_1, t_2, \bullet\bullet\bullet, t_k) \mid t_i \in IT_i\}) \end{aligned} \tag{3.3}$$

where ∇ denotes the optimal downward directed rounding, Δ denotes the optimal upward directed rounding, and ρ denotes the optimal rounding to the nearest machine number.

In (3.3) it is possible to evaluate the endpoints as for ordinary intervals (see for example [17]) and the main values may be computed directly, provided that (a) the computation is carried out with at least as much precision as is used for the endpoints, and (b) the real arithmetic used for the computation rounds the result to the machine number nearest the true result (many computers do not do this).

Another problem is that of out-of-range numbers. We refer the reader to the discussion given in [17]. The only additional difficulty arising in triplex numbers is the inclusion of the main value t_M. Since every improperly computed triplex number should lead to an error message, we must check each of t_L, t_M, and t_R to see whether it was computed successfully or whether one of the errors overflow, infinity, or underflow was encountered. The possible errors are documented in Appendix 3.

4. Using the TRIPLEX arithmetic package:

This section is designed to be the user's guide for the triplex arithmetic package. The package is most easily used with the aid of the AUGMENT precompiler; this use will be described first. Following that, we discuss the use of the package without benefit of the precompiler; essentially, this is just a discussion of how the user should go about doing the things that the precompiler would do were it available.

4.1 Using the package with the AUGMENT precompiler:

Declaring TRIPLEX variables: If X is a TRIPLEX variable, Y is a TRIPLEX vector of length 10, and Z is a 10 x 10 TRIPLEX matrix, the following type declaration will both establish them as TRIPLEX variables and reserve storage for them:

```
TRIPLEX X, Y(10), Z(10,10)
```

Restrictions:

Identifiers should not begin with the prefixes "TPX' or "BPA'; this will help avoid conflicts with the names of subroutines in the package.

If the FORTRAN compiler limits the number of dimensions of an array, that limit must be decreased by 1 for TRIPLEX variables; AUGMENT will declare an undimensioned TRIPLEX variable as a vector.

Options: None

Assigning constant values to TRIPLEX variables: This may be accomplished by a statement which assigns the value of an appropriate Hollerith string to the TRIPLEX variable. For example, to assign the value (.1, .1, .1) to the TRIPLEX variable X, one would write

```
X = 13H(.1, .1, .1)$
```

Restrictions:

The Hollerith string must represent a TRIPLEX as defined in Appendix 1, except that embedded blanks are permitted and will be ignored.

The Hollerith string must be terminated by a field separator: $, #, or = .

The length of the Hollerith string is limited to the length of the formatted read buffer (132 characters in the UNIVAC 1110 version).

The number of significant digits permitted for each endpoint of the triplex number is limited to the value of ESDMAX (60 in the UNIVAC 1110 implementation).

The user should be aware that a statement of the form

```
X = .1
```

will cause a value to be assigned to X , but the resulting interval of the triplex number may not contain .1. The above statement will cause X to be set equal to a degenerate triplex number each of whose endpoints is the REAL approximation to .1; in order to obtain a rigorous triplex number approximation to .1 on most computers, the triplex number would need to be nondegenerate. Thus shortcutting the conversion from Hollerith to triplex can be dangerous.

Options:

Quoted Hollerith literals may be used if the host compiler accepts them.

If the host compiler generates a sentinel for Hollerith literals, and if the TPXUPK primitive recognizes this sentinel (as is the case in the UNIVAC-1110 implementation), the terminal field separator may be omitted.

Implicit conversations from Hollerith to TRIPLEX are allowed (e.g., a Hollerith literal may appear in an arithmetic expression).

Reading TRIPLEX variables -- free format: The free format read will obtain the next data field from the input stream on the specified unit, convert it, and store the resulting value in the specified triplex variable. This may be accomplished by a statement of the form

```
CALL TPXRDF(UNIT,X)
```

Restrictions:

Only one value is read by each call to TPXRDF.

The basic package recognizes units 5 (standard input) and 0 (standard reread).

The Hollerith string read by TPXRDF must represent a triplex number as defined in Appendix 1.

The length of the Hollerith string is limited to the length of the formatted read buffer (132 characters in the UNIVAC 1110 version).

The number of significant digits permitted for each endpoint of the triplex number is limited to the value of ESDMAX (60 in the UNIVAC 1110 version).

The end of an input record does not terminate the input stream.

A call to the free-format read routine with a different unit number specified, or any call to the formatted read routine (see below), will terminate the input stream on the current unit. Once the input stream has been terminated, TPXRDF begins a new input stream with a new record.

TPXRDF will read only from units designated as read units by the information in UNITBL (see "Options" below).

Options:

The standard unit designations may be changed by altering UNITBL, which is located in the COMMON block TPXCCM. (This change would normally be accomplished at the time of adaptation, however.)

Descriptions of additional I/O may be entered in UNITBL. To do this NUNITS must be changed to allow the I/O routines to scan all information in the table (increase NUNITS by 1 for each unit added). Each unit description consists of a row of UNITBL. Column 1 is the unit number; Column 2 is the length of the record (limited to the length of the free format read buffer, which is 132 characters in the 1110 version); Column 3 is the number of characters in each record to be scanned; Column 4 is 0 if the record does not contain a carriage control character and 1 if it does; Column 5 is 1 if the unit is read only, 2 if write only, and 3 if read/write.

Units may be referenced by their position in the table rather than their logical unit numbers: -1 refers to the unit described in the first row of the table, and so on.

The number of characters of each record which are actually scanned may be changed if desired; this allows one to use only the first 72 characters in a card image, for example. Change UNITBL(1,3) to the appropriate value to accomplish this.

The end of a record can be treated as the end of the data field by setting UNITBL(1, 3) to one greater than the length of the physical record and inserting a field termination character in the corresponding character of IBUFRE. The length of this buffer must not be exceeded in doing this, however.

The remainder of the current input record may be skipped by setting OLDUNT to -1 prior to calling TPXRDF.

Each input record may be echoed on the standard print unit as it is read. To accomplish this, set ECHOS to .TRUE. in COMMON block TPXCRD.

Each data field may be echoed on the standard print unit as it is converted. To do this, set ECHOD to .TRUE. in common block TPXCRD.

Delimiters for triplex data may be altered as desired by changing the appropriate value in ICHR, located in COMMON block TPXCCM. Note that this will also affect the Hollerith strings used to assign constant values to triplex variables. If a character is duplicated in this array, its interpretation will be governed by its first appearance.

Data fields may be separated by blanks (as many as desired). Blanks occurring between matching parenteses will be ignored.

Commas within matching pairs of parentheses are regarded as endpoint separators (unless the ICHR array is redefined).

A field may be terminated by

1. A blank which is neither a leading blank nor located between matching parentheses;

2. Any of the characters '$', '#', or '=';

3. A comma occurring outside of matching parentheses;

4. Any nonblank character following a matching right parenthesis; if this is not one of the characters mentioned in (2) or (3) above, then it will be regarded as the first character of the next field.

Any null field or subfield is taken to represent the number 0.

A field containing a single number will be converted to the smallest triplex number containing that number. The resulting triplex number may or may not be degenerate.

If a field contains only two numbers, these are taken to be the endpoints of the triplex numbers and the main value is taken to be the midpoint of the interval thus specified.

Reading TRIPLEX variables -- formatted: The formatted read routine reads and converts a vector of data; the vector may, of course, be of length 1. The user supplies a format, which is an array of length 3: The first element is the number of data fields in each record; the second element is the number of characters to be skipped before beginning the scan on each field; and the third is the number of characters in each data field. The calling sequence is

```
CALL TPXRD (UNIT, FMT, A, N)
```

where UNIT is the unit number, FMT is the format as described above; A is the first location of the vector into which the data is to be read; and N is the length of the vector.

Restrictions:

The basic package recognizes units 5 (standard input) and 0 (reread).

The contents of each field must represent a triplex number as defined in Appendix 1. Field termination characters are not permitted.

The length of each record is limited to the length of the formatted read buffer.

The number of significant digits permitted for each endpoint is limited to the value of ESDMAX.

TPXRD will read only from units designated as read units in UNITBL.

Options:

The standard unit designations may be changed as in TPXRDF.

I/O units may be added to UNITBL; see discussion of TPXRDF.

Units may be referenced by their positions in UNITBL as discussed under TPXRDF.

The number of characters in a record which are actually used may be altered as discussed under TPXRDF.

Input data may be echoed as discussed under TPXRDF.

Delimiters for triplex data may be altered as discussed under TPXRDF.

> Note: Any of the above modifications will affect both of the read routines. Blanks may be embedded in the field; they will be ignored.

The use of parentheses is optional.

A comma will be interpreted as an endpoint separator regardless of whether parentheses are used.

If information is read from a unit on which a carriage control character is indicated, the first character of the record will be ignored.

Computing with TRIPLEX variables: Expressions involving TRIPLEX variables are written in standard FORTRAN syntax, just as though TRIPLEX were a standard FORTRAN data type. A list of the operations and functions available in this package may be found in Appendix 2.

The list of operations and functions available includes all appropriate ANSI standard FORTRAN operations and functions -- i. e., all which have natural triplex extensions. Other operators and functions peculiar to triplex arithmetic and extensions to triplex numbers are implemented; examples include the union of two intervals, the midpoint of an interval, and various measures of the size of an interval. Relational operators are also implemented, but these take on different meanings in the context of triplex arithmetic, and therefore have been given slightly different operator symbols.

Restrictions:

Variable and subprogram names beginning with TPX or BPA should be avoided to preclude conflicts with the package.

If COMMON block declarations are included in the program for the purpose of exercising various options, then the variable names associated with these COMMON blocks must likewise be avoided.

Options:

Mixed mode expressions are permitted, although their use is discouraged due to the high probability of introducing hidden errors. For example, the expression

```
Y = 0.1 * X
```

where X and Y are TRIPLEX variables, will not yield a correct value for Y: 0.1 will first be converted to REAL by the compiler, and AUGMENT will then cause that REAL number to be converted to a degenerate triplex number which will not contain .1 (unless the computer base is decimal-related); multiplication will then take place using the erroneous triplex number. If mixed mode expressions are used, TRIPLEX takes precedence over REAL, INTEGER, and DOUBLE PRECISION (and MULTIPLE, for the multiple precision version of the package) data types. The data type to be used in evaluating each subexpression is determined by normal heierarchy rules.

New special function routines may be added. If this is done, the following conventions and rules should be observed in order to preserve the integrity of the package:

1. The name of the new routine should begin with TPX. Up to three more characters may be appended to form the name of the routine. Conflicts with existing names must be avoided.

2. The calling sequence for the new routine should adhere to the standards of the package: the arguments are listed first, followed by the result. Every effort should be made to minimize the amount of information passed.

3. It is the user's responsibility to insure that the endpoints of the resulting triplex numbers are computed and rounded correctly. All existing package routines are available to the user in writing the new routine, just as they are in writing an applications program.

4. If the new routine is to be invoked by the AUGMENT precompiler, the description deck for AUGMENT must be modified to include the new routine. Consult [5] for details.

5. If errors are possible in the new routine, TPXRAP should be used to handle the error message rather than including an error print- out in the new subprogram. This is done as follows:

    a. The suffix of the name of the new routine is stored in the next available location in the NAMES array.

    b. Appropriate values to indicate the data types of the arguments to the new routine are stored in TYPA, TYPB, and TYPR. Note that it is assumed that the routine will have at most two arguments and one result.

c. The TPXCOM declarations should be included in the new routine. If any argument or result is of a type other than a standard type, BPA, TRIPLEX or EXTENDED, a variable of the appropriate type must be defined and EQUIVALENCEd to TA, TB, or TR respectively; storage of the data in the COMMON block is then done by using the variable name of the appropriate type.

d. TPXRAP should be called with the statement

```
CALL TPXRAP
```

immediately prior to the RETURN statement.

e. Immediately prior to the call on TPXRAP, the values of the arguments must be stored in TA and TB and the value of the result must be stored in TR. TPXFLT must be set to the appropriate fault indicator (If no existinng fault value is approrpirate, a new fault value will need to be defined and TPXRAP will have to be modified to provide the proper printout -- this constitutes a major modification of the package). Finally, the value of ID must be set equal to the index of the name of the new routine in the NAMES array.

Writing TRIPLEX variables: The write routine will convert a vector (possibly of length 1) of triplex variables to external format and write it on the specified unit according to the given format. The format is now an integer array of length 4: the first three elements of the array are the same as for the formatted read, except that unused characters between fields are filled with blanks; FMT(4) is a Hollerith carriage control character (' ' or '0' in the 1110 implementation) for use where approprite. The calling sequence is

```
CALL TPXWR (UNIT, FMT, A, N)
```

where the arguments are as described in the formatted read.

The external representation of each triplex number is guaranteed to contain the triplex number, and is the triplex number with the smallest interval representable in the given format which does so.

Restrictions:

The basic package recognizes units 6 (standard print unit) and 1 (standard punch unit).

The length of each record is limited to the length of the write buffer (132 characters in the UNIVAC 1110 implementation).

The width of each data field specified by the format must be at least great enough to permit the package to convert one significant digit. In the 1110 single precision version, this is 15 digits, assuming a 2-digit exponent. Add two characters for each digit of exponent above 2.

The number of significant digits allowed for each endpoint is limited to the value of ESDMAX (60 in the 1110 implementation).

TPXWR will write only on units designated as write units in UNITBL.

> Note: if any of the restrictions are violated, TPXWR will use a standard format and/or the standard print unit, as necessary, to insure that the designated information is written out in some form.

Options:

The standard unit designations may be changed (see TPXRDF).

I/O units may be added to UNITBL (see TPXRDF).

Units may be referenced by their positions in UNITBL (see TPXRDF).

Delimiters for triplex data may be changed by altering the appropriate values in OCHR, located in COMMON block TPXCCM.

Carriage control specifications may be changed by altering the appropriate element of UNITBL.

Errors: The package is designed to detect all errors as they occur. The TPXRAP routine will print an error message and halt the computation except in those cases where a viable alternative exists. Those cases are few indeed; they comprise arithmetic underflows (where the offending value is set equal to zero or to the properly signed number of smallest magnitude, as approrpriate) and errors occurring on output (where the write routine uses standard modes of output in lieu of the erroneous information). In the former case, computation proceeds without notice to the user; in the latter case, a message is produced on the standard print unit after output is complete.

The possible errors and the response to each are listed in Appendix 3.

Restrictions: None.

Options:

The default response to any fault except 80 (Conversion array overflow) may be changed. To alter the response to Fault number I, change the value of MON(I+1) as desired. This array is located in TPXRCM. MON(80) should never be changed, since this could result in looping under some circumstances.

TPXRAP may be used as a trace routine of sorts by changing the response to a successful operation [MON(1)]. This will cause TPXRAP to produce output even after a successful operation; however, this output is, of course, limited to those routines which call TPXRAP. In cases where errors are not possible, TPXRAP is usually bypassed.

Producing an object program: The generation of an object program is a two-step process.

1. Use AUGMENT to translate the source program into a FORTRAN program compatible with the host FORTRAN compiler. This can be accomplished with a run stream of the following type (see Appendix 6):

```
invoke AUGMENT

[DESCRIPTION decks for TRIPLEX and BPA (see Appendix 5)]

*BEGIN

[source program]

*END
```

   AUGMENT will write the translated program on unit 20. (Note: AUGMENT also uses Unit 21, for scratch work.)

2. Compile the output of AUGMENT using the standard FORTRAN compiler; then load and execute as with any FORTRAN program. The user must insure that the TRIPLEX package library is available to the linkage editor, and that the BLOCK DATA modules are included with the program by the linkage editor.

The run stream for the UNIVAC 1110 version at the University of Wisconsin is shown in Appendix 6.

4.2 Using the package without benefit of AUGMENT:

If necessary, the TRIPLEX package may be used directly, without using AUGMENT to preprocess the source code. If this is done, certain of the things that AUGMENT would normally take care of must be done instead by the user. In this section, we indicate the changes that are necessary in the previous instructions if AUGMENT is not available. These remarks apply primarily to the single precision version of the triplex package. Suitable modification of these instructions will render them applicable to the multiple precision version; however, due to the relative complexity of the multiple precision version, its use in the absence of AUGMENT is discouraged.

Declaring TRIPLEX variables: The number of words of storage that must be reserved for a triplex variable will be three times the number of words needed to store each endpoint. In the UNIVAC 1110 implementation, each triplex variable requires three words of storage. For a particular installation or application, however, the package may have been revised to produce greater accuracy, so the local documentation should be consulted.

The package assumes that the words containing a triplex number will occupy consecutive locations in storage.

The standard data type used to reserve storage for TRIPLEX variables is irrelevant so long as consistency is maintained. AUGMENT has been instructed to declare TRIPLEX variables as REAL arrays of length 3. If the latter convention is used, a dimensioned array of TRIPLEX variables must be declared as a REAL array of one more dimension, the first dimension always being 3.

If X is a TRIPLEX variable, Y is a TRIPLEX vector of length 10, and Z is a 10 x 20 TRIPLEX matrix, they could be declared in the following way:

```
REAL X(3), Y(3, 10), Z(3, 10, 20)
```

Assigning constant values to TRIPLEX variables: This must now be accomplished by a call on the subroutine TPXASG. Depending on the data type used to declare the triplex variables and the whims of the compiler, it may be necessary to refer to, say, X(1) rather than X. The rules for forming the Hollerith strings are the same whether or not AUGMENT is used.

If TRIPLEXes are declared as REAL arrays of length 3, then in order to assign the value of (.1, .1, .1) to X, one would write:

```
CALL TPXASG (13H(.1, .1, .1)$, X(1))
```

Restrictions: No change.

Options: No change.

Reading TRIPLEX variables -- free format: No change, unless the compiler requires explicit subscripts for dimensioned arrays and TRIPLEX variables are declared as arrays. In that case, the subscripts must be furnished.

Reading TRIPLEX variables -- formatted: See comments for free format read.

Computing with TRIPLEX variables: Here, several pitfalls await the unwary user. First, of course, is the necessity of parsing the arithmetic expressions. While this is of vital importance, we assume that the user knows how to do it correctly.

Next is the necessity of expressing triplex variables in a form compatible with the declarations and the restrictions of the compiler. We have touched on this above, and will not iterate here.

Third is the problem of making sure that we are doing exactly what we want to do. For example, if we wish to multiply the TRIPLEX variable X by 2.0, we can not write

```
CALL TPXMUL (X, 2.0, Y)
```

since TPXMUL could multiply X by a "triplex" whose left endpoint was 2.0 and whose main value and right endpoint were whatever happended to by lying around in the next two storage locations. The package, of course, has no way of detecting such blunders; the user alone is responsible for avoiding them.

Thus the user is responsible for employing explicit data type conversions where they are needed, and insuring that all evaluations of expressions are performed using the correct data types.

Restrictions: No change

Options: No change.

Writing TRIPLEX variables: See comments for free format read.

Errors: No change.

Producing an object program: The AUGMENT phase will, of course, not apply. The remaing remarks are valid.

4.3 Using BPA variables in the source program:

The data type of the endpoints of a triplex, and of the results of certain triplex functions, is a nonstandard type named BPA. If BPA format is the same as REAL, computations with these quantities can usually be executed using REAL arithmetic. If a REAL variable is set equal to a BPA expression, AUGMENT will perform an automatic type conversion; if AUGMENT is not used, BPACBR should be called to effect this conversion. If mixed mode expressions involving BPA and standard data types are written, AUGMENT will cause the expression to be evalued using BPA arithmetic, converting the result to a standard data type if necessary.

Computation using BPA routines may be included in the source program, either implicitly, as above, or explicitly. THE USER IS CAUTIONED THAT MOST BPA ROUTINES REQUIRE THAT A ROUNDING OPTION BE STORED IN BPACOM PRIOR TO INVOKING THE ROUTINES (see [17]). Thus, for example, in order to add BPA numbers A and B, using upward-directed rounding, and store the result in C, one would need the following statements in the program:

```
COMMON /BPACOM/ ... (declarations from Appendix 4)
BPA A, B, C
 .
 .
 .
OPTION = RDU
C = A + B
```

If OPTION is not set explicitly, it will have whatever value it may previously have had, and the rounding may not be as expected. This could result in erroneous results.

5. Design, implementation, and adaptation to other hardware:

The current triplex arithmetic package was a straightforward modification of the interval arithmetic package discussed in [17]. In light of the foregoing discussion, it should not be surprising that all that was necessary was to add code to all of the interval routines to compute the main values, adjust formats to accommodate the third component, change the representation of intervals to that of triplexes, and add a few routines to handle the main values.

Since the interval arithmetic package had been written with the aid of AUGMENT, using the nonstandard data type INTERVAL, modification of the package was not at all difficult. We first added a field function MAIN, which either inserts or extracts the main value of the triplex number according as it appears on the left or right side of the = sign. This is known as a field function in the jargon of AUGMENT, and is permitted by AUGMENT whether or not such things are allowed by the FORTRAN compiler. The function MAIN was entirely analogous to the field functions INF and SUP, which we had used in the inplementation of the interval arithmetic package (see Appendix 2).

Having added the MAIN function, it was mechanical to add the lines of code to each routine to compute the main value of the triplex number. For example, in the square root routine, the computation of the main value looks like

```
MAIN(R) = SQRT(MAIN(A))
```

As noted above, errors may occur in computation of the main values as well as in the computation of the endpoints of triplex numbers. We therefore revised the table of faults to accommodate the additional possibilities. The revised table is shown in Appendix 3. This also necessitated some changes to the error-handling routine INTRAP, of course.

Next, we renamed the modules of the interval package by suitable application of a text editing program; we merely replaced the prefix INT with the prefix TPX on each module. Again, since we were using AUGMENT to generate calls on other package modules, there was a minimum of work to do in making these changes -- the most pervasive change was the call to the error- handling routine, which appeared in nearly every routine.

Changing the representation of intervals to that of triplex numbers was very easy. In each routine, interval data had been declared type INTERVAL. We simply changed INTERVAL to TRIPLEX in the package modules, then supplied AUGMENT with a new description deck which defined the structure of triplex numbers.

We did, of course, need to code the functions which implemented the field function MAIN, and revise the primitive functions which had defined intervals in terms of arrays of BPA numbers (such as INF and SUP). These were not difficult tasks.

Perhaps the most difficult of all of the tasks were revision of the input routine to accommodate triplex numbers and alteration of the output formats. Even so, these modifications took less than one day.

Obviously, then, the triplex package described here is very closely related to the interval arithmetic package described in [17]. The design philosophy of the triplex package is identical to that of the interval package, and need not be discussed here. The underlying package for BPA arithmetic is the same package in all particulars, so we refer the reader interested in that part of the triplex package directly to [17].

Nearly all of Section 5 of [17] (Design and implementation of the [interval] package) and Section 6 (adapting the [interval] package to other hardware) transfers directly to the triplex package if the reader mentally makes the same changes in the text that we made in the program: interval to triplex; the prefix INT to the prefix TPX in names of package routines (e.g., INTCOS would become TPXCOS); definition of triplex numbers as BPA arrays of length 3 instead of interval numbers as BPA arrays of length 2. Of course, there are other, less mechanical changes (such as the definition of -T, where T is a triplex number: if T = (a, b, c), then -T := (-c, -b, -a)). The flow chart of a typical triplex function would need to include steps for calculation of the main value. And the state table for the free-format read would need to be revised to admit the possibility of triples instead of pairs. But none of these things are very subtle, and the reader should not have to suffer the indignity of having them spelled out in detail.

6. The multiple precision version of the Triplex Package:

In the foregoing discussion, we have considered primarily the single precision version of the Triplex Package (although the instructions given for use of the package with AUGMENT are equally applicable to the multiple precision version). The single precision version was initially developed for the UNIVAC 1110 computer, but can be transferred to other hardware without a great deal of reprogramming. Indeed, most of the necessary work is in the BPA portion of the package, and this has already been adapted to several other machines. See [18] for details.

The multiple precision arithmetic package of Brent [3] is a completely portable, arbitrary-precision package which includes modules to evaluate all of the standard mathematical functions and some of the nonstandard ones. This package has been interfaced with the AUGMENT precompiler (see [4]).

The multiple precision version of the Triplex Package is based on Brent's multiple precision arithmetic package. The multiple precision package is used as type EXTENDED, and BPA routines have been written for the primitive operations, using an arithmetic format which corresponds quite closely with the format of Brent's multiple precision numbers. The basic architecture of the Triplex Package is, however, unchanged.

In order to use the multiple precision version, one must have access to Brent's package. This may be obtained by writing directly to Brent at the Computing Research Group, Australian National University, Canberra, Australia.

Modification of the single precision version of the triplex package to produce the multiple percision version is entirely analogous to modification of the single precision version of the interval arithmetic package to produce the multiple precision version. The latter process is discussed in detail in [19], and we will not iterate the discussion here.

The basic multiple precision version of the triplex package carries about 28 decimal digits, but this is easily modified by a natural extension of the procedure described in [19].

7. Conclusion:

We have attempted to provide user documentation for the Triplex Package, together with enough technical documentation to assist the user in making modifications that might be desirable in a production environment. The balance of the technical documentation will, as already noted, be found in [17]. Any errors or inadequacies in this document should be reported to the second author.

Since both the package and this document are based on the interval arithmetic package described in [17], those who contributed materially to that effort are, by transitivity, also contributors to this package. They include Dr. F. D. Crary and Dr. S. T. Jones, both formerly of the Mathematics Research Center; Mr. Bill Boyt and Mr. James B. Cheek, of the U. S. Army Corps of Engineers Waterways Experiment Station, Vicksburg, Mississippi; Prof. Bruce D. Shriver of the University of Southwest Louisiana; Prof. Ronnie Ward of the University of Texas at Arlington; Prof. Richard Hetherington of the University of Kansas; and Prof. Myron Ginsberg of Southern Methodist University.

DISCLAIMER

REFERENCES

1. O. Aberth, A precise numerical analysis program, Comm. Assoc. Comput. Mach. 17 (1974), 509-513.

2. K. Boehmer and R. T. Jackson, A FORTRAN-triplex-pre-compiler based on the AUGMENT pre-compiler, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1732, March, 1977.

3. Richard P. Brent, A FORTRAN multiple-precision arithmetic package, Assoc. Comput. Mach. Trans. Math. Software 4 (1978), 57-70.

4. Richard P. Brent, Judith A. Hooper, and J. M. Yohe, An AUGMENT interface for Brent's multiple precision arithmetic package, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1868, August, 1978.

5. F. D. Crary, The Augment precompiler I: User information, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1469, December, 1974 (revised April, 1976).

6. F. D. Crary, The AUGMENT precompiler II: Technical Documentation, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1470, October, 1975.

7. F. D. Crary, A versatile precompiler for nonstandard arithmetics, Assoc. Comput. Mach. Trans. Math. Software (to appear).

8. F. D. Crary and J. M. Yohe, The Augment Precompiler as a tool for the development of nonstandard arithmetic packages, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report (to appear).

9. U. Kulisch, An axiomatic approach to rounded computations, Numer. Math. 18 (1971), 1-17.

10. T. D. Ladner and J. M. Yohe, An interval arithmetic package for the UNIVAC 1108, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1055, May, 1970.

11. Ramon E. Moore, Interval Analysis, Prentice - Hall, Inc., Englewood Cliffs, NJ, 1966.

12. K. Nickel, Triplex-ALGOL and applications, Topics in Interval Analysis (E. Hansen, ed.), Oxford University Press, 1969, 10-24.

13. K. Nickel, Interval-analysis, Proceedings of the Conference on the State of the Art in Numerical Analysis Held at the Univeristy of York April 12 - 15, 1976 (D. Jacobs, ed.), 1977, 193-226.

14. Frederic N. Ris, Tools for the analysis of interval arithmetic, Lecture Notes in Computer Science 29: Interval Mathematics (K. Nickel, ed.) Springer-Verlag, New York, 1975.

15. N.-W. Wipperman et.al., The algorithmic language Triplex-ALGOL-60, Numer. Math. 11 (1968), 175-180.

16. J. M. Yohe, Roundings in floating-point arithmetic, IEEE Trans. Computers C-22 (1973), 577-586.

17. J. M. Yohe, The INTERVAL arithmetic package, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1755, June, 1977 (revised September, 1977).

18. J. M. Yohe, Software for interval arithmetic: a reasonably portable package, Assoc. Comput. Mach. Trans. Math. Software (to appear).

19. J. M. Yohe, The interval arithmetic package - multiple precision version, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report (forthcoming), December, 1978.

APPENDIX 1

STANDARD FORTRAN NUMBER AND TRIPLEX NUMBER REPRESENTATIONS

```
DIGIT       ::= 0|1|2|3|4|5|6|7|8|9

SIGN        ::= +|-

INTEGER     ::= NULL|<SIGN>|<INTEGER><DIGIT>

RADIX       ::= .

FIXEDPOINT  ::= <INTEGER><RADIX>|<FIXEDPOINT><DIGIT>

EXPSEP      ::= E|D

EXPONENT    ::= <SIGN>|<EXPSEP>|<EXPSEP><SIGN>|<EXPONENT><DIGIT>

NUMBER      ::= <INTEGER>|<FIXEDPOINT>|<INTEGER><EXPONENT>|
                <FIXEDPOINT><EXPONENT>

ENDPTSEP    ::= :

COMMA       ::= ,

TRIPLEX     ::= <NUMBER>|(<NUMBER>)|<NUMBER><ENDPTSEP><NUMBER>|
                (<NUMBER><ENDPTSEP><NUMBER>)|(<NUMBER><COMMA><NUMBER>)|
                <NUMBER><ENDPTSEP><NUMBER><ENDPTSEP><NUMBER>|
                (<NUMBER><ENDPTSEP><NUMBER><ENDPTSEP><NUMBER>)|
                (<NUMBER><COMMA><NUMBER><COMMA><NUMBER>)
```

APPENDIX 2

PACKAGE MODULES

This appendix contains information on the modules of the triplex package. The BPA portion of the package is not listed here; the interested reader is instead referred to [17]. The following three pages list the modules of the triplex portion of the package.

The first column of the table gives the function of the module; the second column gives an expanded explanation of the function or, in some cases, the definition of the function. The third column gives the data type of the result of the module, according to the code listed below.

The fourth column gives an example of how the module might be invoked in a program which is to be processed by the AUGMENT precompiler. The reference name of the module (in AUGMENT) is either the function name or the operator shown. Note that all conversion modules may also be invoked implicitly by a replacement operator.

The fifth column indicates how the module would be invoked in the absence of AUGMENT. If the routine is a subroutine (indicated by an 'S' in Column 6), the expression given must be preceded by the word 'CALL'; otherwise, the expression is complete. Note that functions must be typed appropriately in the calling program; Column 6 gives the necessary information in these cases.

Data types: Data types are indicated by one- or two-letter abbreviations. The abbreviations used are as follows:

B - BPA
D - DOUBLE PRECISION
E - EXTENDED
H - HOLLERITH (unpacked)
I - INTEGER
IA - INTEGER ARRAY
L - LOGICAL
PH - PACKED HOLLERITH
R - REAL
X - TRIPLEX

Routine types: S indicates subroutine; any other letter indicates a function of the indicated type (see above). An asterisk denotes a primitive routine.

Variable names: The first letter (or, sometimes, pair of letters) indicates the type of the variable. The last letter is R for result, A, B, or C for argument. Other letters may sometimes be used for special purposes. Explanations not given here will be found in the text or, in cases where the routine is not normally directly accessible to the user, in the program listing.

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	ROUTINE INVOCATION VIA AUGMENT	DIRECT	ROUTINE TYPE
ARITHMETIC					
Add	Sum of two triplexes	X	XA + XB	TPXADD(XA, XB, XR)	S
Subtract	Difference of two triplexes	X	XA - XB	TPXSUB(XA, XB, XR)	S
Multiply	Product of two triplexes	X	XA * XB	TPXMUL(XA, XB, XR)	S
Divide	Quotient of two triplexes	X	XA / XB	TPXDIV(XA, XB, XR)	S
EXPONENTIATION					
to BPA	Raise triplex to BPA power	X	XA ** BB	TPXXXB(XA,BB,XR)	S
to EXTENDED	Raise triplex to EXTENDED power	X	XA ** EB	TPXXXE(XA, EB, XR)	S
to INTEGER	Raise triplex to INTEGER power	X	XA ** IB	TPXXXI(XA, IB, XR	S
to TRIPLEX	Raise triplex to TRIPLEX power	X	XA ** XB	TPXXXX(XA, XB, XR)	S
MATHEMATICAL					
Absolute value	{|x|: x XA}	X	ABS(XA)	TPXABS(XA, XR)	S
Arc cosine	Arc cosine of triplex XA	X	ACOS(XA)	TPXACS(XA, XR)	S
Arc sine	Arc sine of triplex XA	X	ASIN(XA)	TPXASN(XA, XR)	S
Arc tan (2 args)	Arc tangent of XA / XB	X	ATAN2(XA, XB)	TPXAT2(XA, XB, XR)	S
Arc tangent	Arc tangent of triplex XA	X	ATAN(XA)	TPXATN(XA,XR)	S
Cube root	Cube root of triplex XA	X	CBRT(XA)	TPXCBT(XA, XR)	S
Cosine	Cosine of triplex XA	X	COS(XA)	TPXCOS(XA, XR)	S
Hyperbolic cosine	Hyperbolic cosine of triplex XA	X	COSH(XA)	TPXCSH(XA, XR)	S
Exponential	e ** XA	X	EXP(XA)	TPXEXP(XA, XR)	S
Integer	Smallest triplex with integer endpoints containing the triplex XA	X	INT(XA)	TPXINT(XA, XR)	S
Natural logarithm	Log to the base e of triplex XA	X	LN(XA or LOG(XA)	TPXLN(XA, XR)	S
Common logarithm	Log to the base 10 of triplex XA	X	LOG10(XA)	TPXLOG(XA, XR)	S
Sine	Sine of triplex XA	X	SIN(XA)	TPXSIN(XA, XR)	S
Hyperbolic sine	Hyperbolic sine of triplex XA	X	SINH(XA)	TPXSNH(XA, XR)	S
Square root	Square root of triplex XA	X	SQRT(XA)	TPXSQT(XA, XR)	S
Tangent	Tangent of triplex XA	X	TAN(XA)	TPXTAN(XA, XR)	S
Hyperbolic tangent	Hyperbolic tangent of triplex XA	X	TANH(XA)	TPXTNH(XA, XR)	S
FIELD					
Infimum	Left endpoint of triplex XA	B	INF(XA)	TPXINL(BA, XR)(insert)	S
				TPXINF(XA, BR)(extract)	S
Main	Main value of triplex XA	B	MAIN(XA)	TPXMNL(BA, XR)(insert)	S
				TPXMNR(XA, BR)(extract)	S
Supremum	Right endpoint of triplex XA	B	SUP(XA)	TPXSPL(BA, XR)(insert)	S
				TPXSUP(XA, BR)(extract)	S
SPECIAL TRIPLEX FUNCTIONS					
Compose	Form triplex from BPA endpoints and main value	X	COMPOS(BA, BB, BC)	TPXCPS(BA, BB, BC, XR)	S
Distance	Max(Abs(Inf(XA)-Inf(XB)), Abs(Main(XA)-Main(XB) Abs(Sup(XA)-Sup(XB))), rounded up	B	DIST(XA, XB)	TPXDST(XA, XB, BR)	S
Half-length	(Sup(XA)-Inf(XA))/2, rounded up	B	HLGTH(XA)	TPXHLB(XA, BR)	S
Length	Sup(XA)-Inf(XA), rounded up	B	LGTH(XA)	TPXLGB(XA, BR)	S

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	ROUTINE INVOCATION VIA AUGMENT	DIRECT	ROUTINE TYPE
SPECIAL TRIPLEX FUNCTIONS (Continued)					
Magnitude	Sup(Abs(XA))	B	MAG(XA)	TPXMAG(XA, BR)	S
Midpoint	(Sup(XA)+Inf(XA))/2, rounded nearest	B	MDPT(XA)	TPXMDB(XA, BR)	S
Mignitude	Inf(Abs(XA))	B	MIG(XA)	TPXMIG(XA, BR)	S
Pivot	Sqrt(Mag(XA)*Mig(XA)), rounded down	B	PIVL(XA)	TPXPVL(XA, BR)	S
	Same, rounded as specified by IB	B	-	TPXPVO(XA,IB,BR)	S
	Same, rounded up	B	PIVU(XA)	TPXPVU(XA, BR)	S
Intersection	Set-theoretic intersection, with main value taken to be midpoint	X	XA.INTSCT.XB	TPXSCT(XA, XB, XR)	S
Sign	+1 if Inf(XA).GT.0, -1 if Sup(XA).LT.0, 0 otherwise	I	SGN(XA)	TPXSGN(XA)	I
Size	(Abs(Inf(XA)) + Abs(Sup(XA)))/2	B	SIZE(XA)	TPXSIZ(XA, BR)	S
Union	Smallest interval containing both XA, XB main value taken to be arithmetic mean	X	XA.UNION.XB	TPXUNN(XA, XB, XR)	S
CONVERSION					
PH to X	Packed hollerith to triplex	X	CTX(<string>)	TPXASG(HA, XR)	S
E to X	Extended to triplex, bounded at IBth digit	X	-	TPXBND(EA, IB, XR)	S
B to X	BPA to triplex	X	CTX(BA)	TPXCBX(BA, XR)	S
E to X	Extended to triplex	X	CTX(EA)	TPXCEX(EA, XR)	S
UH to X	Unpacked hollerith to triplex(width IB)	X	-	TPXCHX(HA, IB, XR)	S
I to X	Integer to triplex	X	CTX(IA)	TPXCIX(IA, XR)	S
R to X	Real to Triplex	X	CTX(RA)	TPXCRX(RA, XR)	S
X to B	Triplex to BPA	B	CTB(XA)	TPXCXB(XA, BR)	S
X to E	Triplex to extended	E	CTE(XA)	TPXCXE(XA, ER)	D
X to UH	Triplex to unpacked Hollerith(width IB)	UH	-	TPXCXH(XA, HR, IB)	S
X to I	Triplex to integer	I	CTI(XA)	TPXCXI(XA, IR)	I
X to R	Triplex to real	R	CTR(XA)	TPXCXR(XA, RR)	R
SERVICE					
Store	Replacement operator	X	XR = XA	TPXSTR(XA, XR)	S
Negate	Unary minus	X	-XA	TPXNEG(XA, XR)	S
LOGICAL AND RELATIONAL					
Bad triplex	Inf(XA).GT.Main(XA) or Main(XA).GT.Sup(XA)	L	BAD(XA)	TPXBAD(XA)	L
Element of	BA belongs to XB	L	BA .E. XB	TPXELE(BA, XB)	L
Good triplex	Inf(XA).LE.Main(XA).LE.Sup(XA)	L	OK(XA)	TPXOK(XA)	L
Subset of	XA contained in XB	L	XA.SUBSET.XB	TPXSBS(XA, XB)	L

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	ROUTINE INVOCATION VIA AUGMENT	DIRECT	ROUTINE TYPE
LOGICAL AND RELATIONAL (Continued)					
Set-equal	XA componentwise = XB	L	XA .SEQ. XB	TPXSEQ(XA, XB)	L
Set-greater-equal	XA componentwise .GE. XB	L	XA .SGE. XB	TPXSGE(XA, XB)	L
Set-greater	Inf(XA) .GT. Sup(XB)	L	XA .SGT. XB	TPXSGT(XA, XB)	L
Set-less-equal	XA componentwise .LE. XB	L	XA .SLE. XB	TPXSLE(XA, XB)	L
Set-less	Sup(XA) .LT. Inf(XB)	L	XA .SLT.XB	TPXSLT(XA, XB)	L
Set-not-equal	Not set-equal	L	XA .SNE. XB	TPXSNE(XA, XB)	L
Superset	XA contains XB	L	XA.SPRSET.XB	TPXSPS(XA, XB)	L
Value-equal	XA, XB degenerate and equal	L	XA .VEQ. XB	TPXVEQ(XA, XB)	L
Value-greater-eq.	Inf(XA) .GE. Sup(XB)	L	XA .VGE. XB	TPXVGE(XA, XB)	L
Value-greater	Inf(XA) .GT. Sup(XB)	L	XA .VGT. XB	TPXVGT(XA, XB)	L
Value-less-equal	Sup(XA) .LE. Inf(XB)	L	XA .VLE. XB	TPXVLE(XA, XB)	L
Value-less	Sup(XA) .LT. Inf(XB)	L	XA .VLT. XB	TPXVLT(XA, XB)	L
Value-not-equal	XA does not intersect XB	L	XA .VNE. XB	TPXVNE(XA, XB)	L
INPUT/OUTPUT					
Read	Read triplex vector, formatted	X	-	TPXRD(UNIT, FMT, XR, IN)	S
Read,free format	Read next triplex from data stream	X	-	TPXRDF(UNIT, XR)	S
Write	Write triplex vector, formatted	-	-	TPXWR(UNIT, FMT, XA, IN)	S
MISCELLANEOUS					
Reduce	Reduce argument of trig function to principal range	X	-	TPXRED(XA, IB, XR)	S
Unpack	Unpack packed Hollerith	H	-	TPXUPK(HA, HR, IB, IC)	S*
ERROR HANDLING					
Trap	Detect errors in triplex computations (arguments in COMMON)	-	-	TPXRAP	S*

APPENDIX 3

FAULT INFORMATION

FAULT NUMBER	MEANING	ACTION
0	NO FAULT	0
1-63	ARITHMETIC FAULTS IN TRIPLEX NUMBER: FAULT CODE IS 16*FAULT(L)+4*FAULT(M)+FAULT(R) WHERE L=LEFT, M=MAIN, AND R=RIGHT, AND FAULT CAN HAVE ANY OF THE FOLLOWING VALUES: 0 - SUCCESSFUL OPERATION 1 - OVERFLOW 2 - INFINITY 3 - UNDERFLOW FOR EXAMPLE, A FAULT VALUE OF 43 = 2*16 + 2*4 + 3 WOULD MEAN THAT THE LEFT ENDPOINT AND MAIN VALUE WERE BOTH (MINUS) INFINITY, AND THE RIGHT ENDPOINT UNDERFLOWED. THE ACTION CODES FOR THESE FAULTS ARE SET AS FOLLOWS: LOGICALLY IMPOSSIBLE FAULTS ANY FAULT INVOLVING INFINITY WHICH IS LOGICALLY POSSIBLE ANY FAULT INVOLVING ONLY SUCCESSFUL OPERATIONS OR UNDERFLOWS	 4 3 0
64	DIVISION BY ZERO	3
65	ZERO ** ZERO	1
66	SQUARE ROOT OF NEGATIVE NUMBER	3
67	LOG OF NONPOSITIVE NUMBER	3
68	UNDERFLOW DURING BPA COMPUTATION	0
69	OVERFLOW DURING BPA COMPUTATION	3
70	INTERSECTION OF DISJOINT TRIPLEXES	3
71	ARC COS OR ARC SIN ARGUMENT OUT OF RANGE	3
72	INVERTED TRIPLEX	4
73	ILLEGAL INPUT CHARACTER	4
74	ILLEGAL INPUT FORMAT SPECIFICATION	4
75	ILLEGAL OUTPUT FORMAT SPECIFICATION	1
76	INPUT STRING TOO LONG	4
77	UNSPECIFIED INPUT UNIT	4
78	END OF FILE ON INPUT	1
79	UNSPECIFIED OUTPUT UNIT	1
80	CONVERSION ARRAY OVERFLOW	4(A)
81	UNRECOGNIZED ERROR	4

(A) THIS FAULT VALUE SHOULD NOT BE CHANGED, SINCE ANY OTHER ACTION COULD RESULT IN A RECURSIVE CALL ON TPXRAP FROM TPXCXH.

TPXRAP WILL TAKE ONE OF SEVERAL POSSIBLE ACTIONS, DEPENDING ON THE VALUE OF THE CELL IN THE MON ARRAY CORRESPONDING TO THE FAULT WHICH HAS OCCURRED. POSSIBLE VALUES AND CORRESPONDING ACTIONS ARE

0	RETURN WITHOUT TAKING ACTION
1	PRINT MESSAGE
2	PRINT MESSAGE AND TRACE CALL SEQUENCE
3	AS (2), AND STEP ERROR COUNTER
4	AS (2), BUT THEN TERMINATE JOB

IF NO INFORMATION TO THE CONTRARY IS SUPPLIED, TPXRAP WILL ASSUME THAT THE CALLING ROUTINE IS A SUBROUTINE WITH TWO TRIPLEX ARGUMENTS -- AN OPERAND AND A RESULT.

APPENDIX 4

COMMON BLOCK INFORMATION

This appendix contains information concerning the COMMON blocks used in the Triplex Package. This is an abbreviated version of the corresponding appendix in [$Y1], and the reader needing more information should consult that document. The changes indicated in Section 5 will render that appendix valid for the Triplex package.

Here, we list only the COMMON block declarations, together with DATA statements for the portions of the Triplex Package which are not dependent on the representation of BPA, EXTENDED, or TRIPLEX. Thus, the reader can see what declarations are necessary and the values assigned to the variables in question. This should enable one to make any modifications that would be desirable in any routine use of the package.

```
1.        COMMON-BLOCK FOR DEFINING ROUNDING OPTIONS AND PASSING
C         SELECTED ROUNDING OPTION, ACCURACY, AND RESULTING FAULTS TO
C         AND FROM THE BPA PACKAGE
      COMMON /BPACOM/ OPTION, BPAFLT, ACC, RDU, RDL, RDT, RDN, RDA
      INTEGER OPTION, BPAFLT, ACC, RDU, RDL, RDT, RDN, RDA
      DATA RDU /1/, RDL /2/, RDT /3/, RDN /4/, RDA /5/

2.        COMMON BLOCK NEEDED TO HOLD PARAMETERS FOR BPA ROUTINES.
C         THESE CONSTANTS ARE MACHINE - DEPENDENT.
      COMMON /BPACON/ ZRO, ONE, ONM, TWO, PIO2, PI, BPAMNB, BPAMXB
      BPA             ZRO, ONE, ONM, TWO, PIO2, PI, BPAMNB, BPAMXB
      DATA ZRO    /<BPA representation of 0>/,
     1     ONE    /<BPA representation of 1>/,
     2     ONM    /<BPA representation of -1>/,
     3     2    /<BPA representation of TWO>/,
     4     PIO2   /<BPA representation of PI/2, rounded up>/,
     5     PI     /<BPA representation of PI, rounded up>/,
     6     BPAMNB /<BPA representation of smallest positive BPA number>/
     7     BPAMXB /<BPA representation of largest positive BPA number>/

3.        COMMON BLOCK FOR CONSTANTS DEALING WITH ACCURACY OF
          LIBRARY FUNCTIONS
      COMMON /BPAACC/ IACC(24), LNACC, LGACC, SNHACC,TNHACC, EXPMXA,
     1                EXPMNA, FRACBD, MAXINT
      INTEGER IACC
      BPA LNACC, LGACC, SNHACC, TNHACC, EXPMXA, EXPMNA, FRACBD, MAXINT
C         VALUES ARE GIVEN FOR ACCURACY (IN BITS) OF THE EXTENDED
C         LIBRARY FUNCTIONS  SIN, COS, TAN, ASN, ACS, ATN, LN(AWAY FROM
C         1), LN(NEAR 1), LOG10(AWAY FROM 1), LOG10(NEAR 1), EXP,
C         SNH(AWAY FROM 0), SNH(NEAR 0), COSH, TANH(AWAY FROM 0),
C         TANH(NEAR 0), SQRT, AND CBRT, RESPECTIVELY.
      DATA IACC /<values for accuracy in digits of individual routines, as
     1           listed above/
      DATA LNACC  /<radius of nbd of 1 in which LN is less accurate>/,
     1     LGACC  /<radius of nbd of 1 in which LOG10 is less accurate>/,
```

```
     2     SNHACC /<radius of nbd of 0 in which SINH is less accurate>/,
     3     TNHACC /<radius of nbd of 0 in which TANH is less accurate>/,
     4     EXPMXA /<largest BPA such that EXP(EXPMXA) does not overflow>/,
     5     EXPMNA /<smallest BPA such that EXP(EXPMNA) does not underflow>/,
     6     FRACBD /<smallest BPA number with all digits preceding radix>/,
     7     MAXINT /<BPA representation of (largest integer + 1)>/

4.        COMMON BLOCK FOR PASSING INFORMATION NEEDED BY BPA
C         CONVERSION ROUTINES
      COMMON /BPACCM/ EXWDTH, ESDMAX, EXMAX, EXMIN,
     1                IX(80), ICX(40), ECX(40), EX(80)
      INTEGER         EXWDTH, ESDMAX, EXMAX, EXMIN,
     1                IX, ICX, ECX, EX
      DATA EXWDTH /2/, ESDMAX /60/, EXMAX /40/, EXMIN /-40/
      DATA IX  /80, 2, 1,12, 0, 0, 0, 0, 27/,
     1     ICX /40, 65536, 16, 21, 0, 0, 0, 0, 3/,
     2     ECX /40, 100000, 5, 21, 0, 0, 0/,
     3     EX  /80, 10, 1,12/

5.        COMMON BLOCK FOR COMMUNICATION WITH ERROR-HANDLING ROUTINE
      COMMON /TPXCOM/ TPXFLT, ID, TA, TB, TR
      INTEGER TPXFLT, ID
      TRIPLEX TA, TB, TR
      EXTENDED TAE, TBE, TRE
      EQUIVALENCE (TA, TAE), (TB, TBE), (TR, TRE)

6.        MACHINE DEPENDENT TRIPLEX CONSTANTS AND EXTENDED CONSTANTS
          FOR REDUCTION OF TRIG FUNCTION ARGUMENTS TO PRINCIPAL RANGE
      COMMON /TPXCON/ PIO2I, PII, TPIO2I, TPII, NIN8TH,
     1                EPI, ETPI, DELTA, EONE
      TRIPLEX         PIO2I, PII, TPIO2I, TPII, NIN8TH
      EXTENDED         EPI, ETPI, DELTA, EONE
      REAL REPI(2), RETPI(2), RDELTA(2), REONE(2)
      EQUIVALENCE(REPI(1), EPI), (RETPI(1), ETPI), (RDELTA(1), DELTA),
     1           (REONE(1), EONE)
      DATA PIO2I  /<triplex representation of PI/2>/,
     1     PII    /<triplex representation of PI>/,
     2     TPIO2I /<triplex representation of 3PI/2>/,
     3     TPII   /<triplex representation of 2PI>/,
     4     NIN8TH /<triplex representation of 9/8>/,
     5    REPI    /<extended representation of PI>/,
     6    RETPI   /<extended representation of 2PI>/,
     7    RDELTA  /<precision of extended arithmetic>/,
     8    REONE   /<extended representation of 1.0>/

7.        CONSTANTS FOR TRIPLEX I/O ROUTINES
      COMMON /TPXCCM/ NUNITS, UNITBL(8, 5), NOCHR, ICHR(10), OCHR(4),
     1       LRBFOR, RBFORM(132), LRBFRE, RBFREE(132), IRBFRE, OLDUNT,
     2       LWBFOR, WBFORM(132), SOFMT(4), IWMIN, NOCCC, CCC(2),
     3       KFMT(4), HSTR(132)
      INTEGER NUNITS, UNITBL, NOCHR, ICHR, OCHR, LRBFOR, RBFORM, LRBFRE,
     1        RBFREE, IRBFRE, OLDUNT, LWBFOR, WBFORM, SOFMT, IWMIN,
     2        NOCCC, CCC, KFMT, HSTR
```

```
      DATA NUNITS /4/, ((UNITBL(I, J), J = 1, 5), I = 1, 4)
     1     / 5,  80,  80, 0, 1,
     2       6, 132, 132, 1, 2,
     3       1,  80,  80, 0, 2,
     4       0,  80,  80, 0, 1/
      DATA NOCHR /10/
      DATA ICHR /1H(, 1H:, 1H), 1H , 1H,, 1H#, 1H=, 1H$, 1H0, 1H /
      DATA OCHR /1H(, 1H,, 1H), 1H /
      DATA IBUFRE /133/, OLDUNT /-1/, LRBFOR /132/, LRBFRE /132/,
     1     LWBFOR /132/, SOFMT /2, 1, 55, 1H /, IWMIN /22/, NOCCC /2/,
     2     CCC /1H , 1H0/

8.          COMMON BLOCK TO CONTROL ECHOING OF INPUT RECORDS AND FIELDS
      COMMON /TPXCRD/ ECHOS, ECHOD
      LOGICAL ECHOS, ECHOD
      DATA ECHOS /.FALSE./, ECHOD /.FALSE./

9.          COMMON BLOCK DECLARATIONS FOR TPXRAP
              THE PURPOSE OF THIS COMMON BLOCK IS TO GIVE THE USER
              ACCESS TO THE PARAMETERS USED BY TPXRAP WITHOUT REQUIRING
              RECOMPILATION OF TPXRAP.
      COMMON /TPXRCM/ MON(88), NAMES(40), TYPA(40), TYPB(40), TYPR(40)
      INTEGER MON, NAMES, TYPA, TYPB, TYPR
      DATA MON /0, 4, 3, 0, 4, 4, 3, 4, 4, 4, 3, 4, 0, 4, 3, 0,
     1          4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4,
     2          3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 3, 3,
     3          0, 4, 3, 0, 4, 4, 3, 4, 4, 4, 3, 4, 0, 4, 3, 0,
     4          3, 1, 3, 3, 0, 3, 3, 3, 4, 4, 4, 1, 4, 4, 1, 1, 4, 4/
      DATA NAMES /3HADD, 3HSUB, 3HMUL, 3HDIV, 3HSCT, 3HXXI, 3HSIN,
     1            3HCOS, 3HTAN, 3HASN, 3HACS, 3HATN, 3HEXP, 3HLN ,
     2            3HLOG, 3HSNH, 3HCSH, 3HTNH, 3HSQT, 3HCBT, 3HMDB,
     3            3HHLB, 3HLGB, 3HCXI, 3HBND, 3HCEX, 3HRD , 3HRDF,
     4            3HASG, 3HCHX, 3HWR , 3HCXH, 3HCRX, 3HDST, 3HCPS/
      DATA TYPA /6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
     1           6, 6, 6, 6, 6, 6, 5, 5, 0, 0, 0, 0, 0, 0, 2, 6, 0/
      DATA TYPB /6, 6, 6, 6, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1           0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 6, 0/
      DATA TYPR /6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
     1           6, 6, 4, 4, 4, 1, 6, 6, 0, 0, 0, 6, 0, 6, 6, 4, 6/
```

APPENDIX 5

TRIPLEX DESCRIPTION DECKS FOR AUGMENT

SINGLE PRECISION VERSION OF TRIPLEX PACKAGE

```
*DESCRIBE EXTENDED                                                      DST00010
DECLARE DOUBLE PRECISION, KIND FUNCTION, PREFIX EXT                     DST00020
COMMENT DUMMY DESCRIPTION SO CONVERT CAN BE USED                        DST00030
FUNCTION INT (=DINT(STD), (DOUBLE PRECISION), DOUBLE PRECISION)         DST00040
*DESCRIBE BPA                                                           DST00050
DECLARE REAL, KIND SAFE SUBROUTINE, PREFIX BPA                          DST00060
OPERATOR +  (, NULL UNARY, PRV, $)                                      DST00070
COMMENT  THE FOLLOWING DESCRIPTOR MUST BE INSERTED IN THE DECK AT THIS  DST00080
 POINT IF THE REAL UNARY MINUS OF FORTRAN DOES NOT SUFFICE FOR THE BPA  DST00090
 UNARY MINUS --                                                         DST00100
 OPERATOR -  (NEG, UNARY, PRV, $)                                       DST00110
OPERATOR +  (ADD, BINARY 1, PRV, $, COMM), * (MUL), - (SUB,,,,NONCOMM), DST00120
        /  (DIV)                                                        DST00130
OPERATOR ** (XBI, BINARY 3, PRV, $, INTEGER, $)                         DST00140
OPERATOR .EQ. (EQ, BINARY 2, PRV, $, LOGICAL), .NE. (NE), .LT. (LT),    DST00150
        .LE. (LE), .GT. (GT), .GE. (GE)                                 DST00160
FUNCTION ABS (ABS, ($), $), SIN (SIN), COS (COS), TAN (TAN), ASIN (ASN),DST00170
        ACOS (ACS), ATAN (ATN), LOG10 (LOG), LOG (LN), LN (),           DST00180
        EXP (EXP), SINH (SNH), COSH (CSH), TANH (TNH), SQRT (SQT),      DST00190
        CBRT (CBT)                                                      DST00200
FUNCTION INT (INT, ($), $)                                              DST00210
FUNCTION MAX (MAX, ($, $), $), MIN (MIN)                                DST00220
FUNCTION SGN (SGN, ($), INTEGER)                                        DST00230
COMMENT  THE FOLLOWING DESCRIPTOR MUST BE INSERTED IN THE DECK AT THIS  DST00240
 POINT IF THE REAL REPLACEMENT OPERATOR IN FORTRAN WILL NOT SERVE AS THEDST00250
 BPA REPLACEMENT OPERATOR                                               DST00260
 SERVICE COPY(STR)                                                      DST00270
CONVERSION CTB (CEB, DOUBLE PRECISION, $, DOWNWARD),                    DST00280
          CTB (CRB, REAL, $, UPWARD),                                   DST00290
          CTB (CIB, INTEGER, $, UPWARD),                                DST00300
          CTE (CBE, $, DOUBLE PRECISION, UPWARD),                       DST00310
          CTR (CBR, $, REAL, DOWNWARD),                                 DST00320
          CTI (CBI, $, INTEGER, DOWNWARD)                               DST00330
COMMENT  THE FOLLOWING TWO CARDS ARE USED TO FORCE AUGMENT TO USE THE   DST00340
 REAL UNARY MINUS OPERATOR OF FORTRAN AS THE BPA UNARY MINUS OPERATOR --DST00350
*ENVIRONMENT                                                            DST00360
OPERATOR -  (UNARY, PRV, BPA)                                           DST00370
*DESCRIBE TRIPLEX                                                       DST00380
COMMENT  IN THIS DESCRIPTION DECK, 'DOUBLE PRECISION' IS USED AS A      DST00390
 SYNONYM FOR 'EXTENDED'.  IF EXTENDED IS A TYPE OTHER THAN DOUBLE PRE-  DST00400
 CISION, THE APPROPRIATE CHANGES WILL NEED TO BE MADE IN THIS DECK.     DST00410
DECLARE REAL(3), KIND SAFE SUBROUTINE, PREFIX TPX                       DST00420
OPERATOR +  (,NULL UNARY, PRV, $)                                       DST00430
OPERATOR -  (NEG, UNARY, PRV, $)                                        DST00440
OPERATOR +  (ADD, BINARY1, PRV, $, COMM), * (MUL), - (SUB,,,,NONCOMM),  DST00450
        /  (DIV)                                                        DST00460
```

```
OPERATOR ** (XXI, BINARY 3, PRV, $, INTEGER, $), ** (XXB,,,,BPA),          DST00470
         ** (XXE,,,,DOUBLE PRECISION), ** (XXX,,,,$)                       DST00480
OPERATOR .VEQ. (VEQ, BINARY 2, .EQ. , $, LOGICAL, COMM), .VNE. (VNE),      DST00490
         .SEQ. (SEQ), .SNE. (SNE)                                          DST00500
OPERATOR .VLT. (VLT, BINARY 2, .LT. , $, LOGICAL), .VLE. (VLE),            DST00510
         .VGT. (VGT), .VGE. (VGE), .SLT. (SLT), .SLE. (SLE),               DST00520
         .SGT. (SGT), .SGE. (SGE)                                          DST00530
OPERATOR .UNION. (UNN, BINARY 1, .OR., $), .INTSCT. (SCT,, .AND.)          DST00540
OPERATOR .SUBSET. (SBS, BINARY 2, .EQ., $, LOGICAL), .SPRSET. (SPS)        DST00550
OPERATOR .E. (ELE, BINARY 3, .EQ., BPA, $, LOGICAL)                        DST00560
FUNCTION ABS (ABS, ($), $), SIN(SIN), COS (COS), TAN (TAN), ASIN (ASN),    DST00570
         ACOS (ACS), ATAN (ATN), LOG10 (LOG), LN (LN), EXP (EXP),          DST00580
         SINH (SNH), COSH (CSH), TANH (TNH), SQRT (SQT), CBRT(CBT),        DST00590
         LOG (LN)                                                          DST00600
FUNCTION ATAN2 (AT2, ($, $), $)                                            DST00610
FUNCTION OK  (OK, ($), LOGICAL), BAD (BAD)                                 DST00620
FUNCTION COMPOS (CPS, (BPA, BPA, BPA), $)                                  DST00630
FUNCTION INT (INT, ($), $)                                                 DST00640
FUNCTION DIST (DST, ($, $), BPA)                                           DST00650
FUNCTION MDPT (MDB, ($), BPA), HLGTH (HLB), LENGTH (LGB), LGTH (LGB),      DST00660
         MAG (MAG), MIG (MIG), PIVL (PVL), PIVU (PVU), SIZE (SIZ)          DST00670
FUNCTION SGN (SGN, ($), INTEGER)                                           DST00680
FUNCTION BOUND (BND, (DOUBLE PRECISION, INTEGER), $)                       DST00690
FIELD SUP (SUP, SPL, ($), BPA), INF (INF, INL), MAIN (MNR, MNL)            DST00700
SERVICE COPY (STR)                                                         DST00710
CONVERSION CTX (CIX, INTEGER, $, UPWARD),                                  DST00720
           CTX (CBX, BPA, $, UPWARD),                                      DST00730
           CTX (CRX, REAL, $, UPWARD),                                     DST00740
           CTX (CEX, DOUBLE PRECISION, $, UPWARD),                         DST00750
           CTX (ASG, HOLLERITH, $, UPWARD),                                DST00760
           CTI (CXI, $, INTEGER, DOWNWARD),                                DST00770
           CTB (CXB, $, BPA, DOWNWARD),                                    DST00780
           CTR (CXR, $, REAL, DOWNWARD),                                   DST00790
           CTE (CXE, $, DOUBLE PRECISION, DOWNWARD)                        DST00800
```

APPENDIX 6

RUNSTREAM FOR USE OF THE TRIPLEX PACKAGE WITH AUGMENT

We first give an outline of the AUGMENT runstream which should be applicable to and AUGMENT installation.

```
1. [ATTACH APPROPRIATE FILES TO OBTAIN AUGMENT AND
       THE DESIRED NONSTANDARD PACKAGES]
2. [INVOKE AUGMENT]
   PRINT SUPPRESS (if listing of the description decks
                   is not desired)
3. [SUPPLY APPROPRIATE DESCRIPTION DECKS TO AUGMENT]
   *BEGIN
   <any special cards>
   *DISABLE WARNINGS (if AUGMENT warning messages
                                 are not desired)
4. [SOURCE CODE TO BE PROCESSED BY AUGMENT]
   *END
5. [COMPILE PREPROCESSED PROGRAM, WHICH HAS BEEN WRITTEN
      ON LOGICAL UNIT 20]
6. <any other processing, such as preparation of files, compi-
    lation of other modules, etc.>
7. [INVOKE LINKAGE EDITOR]
   <special instructions to linkage editor, aside from
     the rather obvious requirement that the nonstandard
     package must be loaded with the program>
8. [EXECUTE PROGRAM]
   <data>
9. [END OF JOB]
```

The following runstream for the single precision version of the Triplex Package on the UNIVAC 1110 installation at the University of Wisconsin - Madison shows how the above runstream is implemented in a specific situation:

```
@ASG,A MRC*LIB.
@USE A.,MRC*LIB.
@ASG,A MRC*TRIPLEX.
@USE P.,MRC*TRIPLEX.
@XQT A.AUGMENT
PRINT SUPPRESS (if listing of the description decks
                is not desired)
@ADD A.DESCRIPTION/FORTRAN-V
@ADD P.DESCRIPTION/TRIPLEX
*BEGIN
*CONVERT EXTENDED - DOUBLE PRECISION
*DISABLE WARNINGS (if AUGMENT warning messages
                                 are not desired)
:FOR,<options> <filename,etc.>  (note ":" rather than
                                  "@" in Column 1)
<program>
:FOR,etc.
<subprogram>
```

```
etc.
*END
@ADD 20.
<any other processing, such as @PREP of files, compi-
 lation of other modules, etc.>
@MAP,I <absolute element>
 IN TPF$.
 IN A.BPACOM
 IN P.TPXCCOM
 LIB A.
 LIB P.
@XQT <absolute element>
<data>
@FIN
```

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: #1901
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): TRIPLEX ARITHMETIC FOR FORTRAN
5. TYPE OF REPORT & PERIOD COVERED: Technical Summary Report, no specific reporting period
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Klaus Boehmer and J. M. Yohe
8. CONTRACT OR GRANT NUMBER(s): DAAG29-75-C-0024, BO 622/1

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, Wisconsin 53706
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 8 - Computer Science

11. CONTROLLING OFFICE NAME AND ADDRESS: See Item 18
12. REPORT DATE: December 1978
13. NUMBER OF PAGES: 38

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 43p.
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

MRC-TSR-1901

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

U. S. Army Research Office
P.O. Box 12211
Research Triangle Park,
North Carolina 27709

Institut fuer Praktische Mathematik
Universitaet Karlsruhe
D-7500 Karlsruhe, Germany

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interval Analysis
Interval Arithmetic
Multiple Precision
Extended Precision
Portable Software
Software Package
Precompiler Interface
Augment Interface
Triplex Arithmetic

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Triplex arithmetic is a variant of interval arithmetic in which a "main" value is computed in addition to the endpoints of the containing interval. The "main" value is, in general, the value that would have been computed had properly-rounded real arithmetic been used to compute the results of the calculation; it may, in some sense, be regarded as the "most probable" value of the result of the calculation. The endpoints of the interval define the worst-case range of the possible values of the computation.

→ next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

221 220

Abstract (continued)

This report describes a FORTRAN implementation of triplex arithmetic in both single and multiple precision. The package described in this report is designed to be used with the AUGMENT precompiler; this makes triplex arithmetic particularly easy to use.